Bab 6

Query (I): Dasar-Dasar SELECT

Query adalah inti dari SQL. Q pada SQL adalah singkatan dari "query". SQL diciptakan dengan tujuan utama semata-mata agar kita bisa melakukan query pada database dengan lebih mudah, tanpa pemrograman dan tanpa harus menguasai istilah dan simbol matematika yang njlimet. Kemampuan membuat query yang tepat dan efisien, mulai dari yang sederhana hingga yang kompleks, merupakan patokan utama untuk mengatakan bahwa seseorang telah menguasai bahasa SQL. Jangan heran pula kalau saya katakan bahwa bab-bab sebelum bab ini sebetulnya tak lebih dari *prelude*, sajian pembuka, bagi hidangan utama pada buku ini yaitu mengenai query. Memang SQL tidak hanya sekedar query dan Anda perlu mengetahui juga mengenai cara membuat dan mengubah tabel, impor/ekspor data, transaksi, dan lain sebagainya. Tapi barangkali sekitar 80% ilmu SQL ada pada query. Lihat saja manual MySQL atau database lain. Keterangan mengenai perintah INSERT, UPDATE, DELETE, dan lain sebagainya mungkin masing-masing hanya beberapa halaman. Sintaksnya pun cukup sederhana. Tapi untuk perintah SELECT, yaitu untuk melakukan query, sintaksnya amat beragam dan kompleks, dan penjelasan mengenai SELECT bisa memakan tempat banyak halaman.

Query itu setengah ilmu dan setengah seni. Sebuah query yang sama seringkali bisa ditulis dalam beberapa cara, dan memilih mana yang paling efisien bergantung pada seperti apa data Anda maupun apa produk database yang Anda gunakan. Kadang kita bisa menulis query yang singkat dan efisien, namun menggunakan fitur spesifik sebuah database sehingga tidak bisa dipakai di produk lain. Seringkali pula query berhubungan erat dengan desain tabel database, sehingga untuk mempermudah sebuah query kadang dibutuhkan perubahan atau perbaikan pada skema databasenya.

Contoh yang banyak, latihan yang sering, ditambah pengalaman merupakan tiga kunci menguasai ilmu dan seni query SQL. Di buku ini, saya akan membagi pembahasan mengenai query ke dalam tiga bab. Bab yang pertama akan mengajarkan dasar-dasar SELECT untuk tabel tunggal. Bab query yang kedua akan mengajarkan tentang query multitabel menggunakan JOIN dan subquery. Bab query yang ketiga membahas topik-topik lain seperti tree dan optimisasi. Setelah itu, menyusul bab tentang contoh-contoh aplikasi. Diharapkan dengan contoh-contoh tersebut Anda memiliki gambaran yang lebih jelas mengenai berbagai macam teknik query. Setelah itu bergantung pada Anda untuk berlatih dan menimba pengalaman agar sepenuhnya menguasai SQL.

6.1 Berkenalan Dengan SELECT

SELECT adalah perintah utama di SQL untuk melakukan query (atau bahkan bisa dibilang perintah satu-satunya; bergantung pada seluas apa definisi "query"). SELECT adalah swiss-army knife yang memiliki banyak klausa dan seluk-beluk sehingga di buku ini tidak akan dijelaskan sekaligus sintaksnya melainkan sepotong demi sepotong.

Sintaks dasar perintah SELECT:

```
-- Resep 6-1-1: Sintaks dasar perintah SELECT
SELECT ekspresi_select [FROM relasi[,...]]
  [klausa WHERE]
  [klausa GROUP BY]
  [klausa HAVING]
  [klausa ORDER BY]
  [klausa LIMIT]
```

Perintah SELECT mengambil data dan/atau mengkalkulasi data di dalam *ekspresi_select*. Data ini diambil dari satu atau lebih *relasi*. *Relasi* dapat berupa tabel atau view atau perintah SELECT lain. *Klausa WHERE* untuk memilih baris-baris

tertentu saja dari *relasi*. *Klausa GROUP BY* digunakan untuk mengumpulkan beberapa baris dari relasi menjadi hanya satu baris saja, dan digunakan dalam perhitungan total, rata-rata, atau statistik. *Klausa HAVING* mirip seperti *klausa WHERE* yaitu untuk memilih baris-baris tertentu saja, tapi baris-baris tertentu hasil dari GROUP BY. *Klausa ORDER BY* untuk mengurutkan baris-baris hasil. *Klausa LIMIT* untuk membatasi hanya mengambil sekian baris pertama atau baris kesekian hingga baris kesekian.

Hasil perintah SELECT adalah nol atau lebih baris-baris, dengan tiap baris berisi kolom-kolom dengan nama dan urutan sesuai yang didefinisikan di *ekspresi_select*.

Setiap anak kalimat (klausa) nanti akan dibahas dalam tiap subbab tersendiri.

Mengambil Semua Baris Dari Tabel

```
-- Resep 6-1-2: Mengambil Semua Baris Dari Tabel
SELECT * FROM nama_tabel;
```

Bentuk perintah ini adalah salah satu yang paling sering dipakai (dan telah beberapa kali kita lihat pada contoh di bab-bab sebelumnya). Pada dasarnya query ini berarti "tampilkan seluruh isi tabel". Tidak ada klausa WHERE, klausa ORDER BY, ataupun klausa-klausa lainnya. Ekspresi SELECT di sini adalah * yang artinya untuk mewakili semua kolom yang ada pada tabel, dengan urutan yang sama sesuai waktu CREATE TABLE (namun perhatikan, sebetulnya standar SQL tidak mengatur bahwa urutan harus sama, sehingga mungkin di produk database lain urutannya akan acak dan Anda tidak boleh menganggap bahwa urutannya selalu sama).

Urutan baris tampil pun tidak bisa dijamin di sini, meskipun rata-rata database umumnya menampilkan baris-baris sesuai urutan kronologis INSERT.

Sebuah contoh:

```
CREATE TABLE orang (
  id INT PRIMARY KEY,
  nama_depan VARCHAR(32),
  nama_belakang VARCHAR(64),
  tanggal_lahir DATE
);
INSERT INTO orang VALUES (1, 'Andi', 'Firmansyah', '1988-10-1');
INSERT INTO orang VALUES (2, 'Rieke', 'Ardianti', '1968-12-28');
```

```
INSERT INTO orang VALUES (3, 'Tukiman', '' , '1981-11-1');
INSERT INTO orang VALUES (4, 'Budiman', 'Ali', '1981-4-3');
INSERT INTO orang VALUES (5, 'Andi', 'Simanjuntak', '1975-6-23');

 mysql> SELECT * FROM orang;
+----+------------+--------------+--------------+
| id | nama_depan | nama_belakang | tanggal_lahir |
+----+------------+--------------+--------------+
|  1 | Andi       | Firmansyah    | 1988-10-01    |
|  2 | Rieke      | Ardianti      | 1968-12-28    |
|  3 | Tukiman    |               | 1981-11-01    |
|  4 | Budiman    | Ali           | 1981-04-03    |
|  5 | Andi       | Simanjuntak   | 1975-06-23    |
+----+------------+--------------+--------------+
5 rows in set (0.05 sec)
```

Bisa dilihat bahwa perintah SELECT yang diberikan di klien console mysql ini mengembalikan isi tabel dengan urutan kolom sesuai dengan urutan sewaktu CREATE TABLE, dan urutan baris sesuai dengan urutan INSERT.

Di bahasa pemrograman PHP, pada umumnya hasil dari query diambil dengan fungsi `mysql_fetch_row()` atau `mysql_fetch_assoc()`. Jika query kita berupa "SELECT * ..." maka saya sarankan agar membiasakan menggunakan varian `-assoc()` untuk menampung ke array asosiatif, bukan array berindeks numerik. Ini karena, seperti telah disebutkan bahwa standar SQL tidak menjamin urutan kolom sesuai dengan urutan CREATE TABLE. Walaupun MySQL mungkin menjaminnya, namun produk database lain mungkin tidak; jadi kebiasaan yang baik dan portabel adalah untuk tidak berasumsi bahwa urutan selalu dijamin. Lagipula, jika sudah terjadi beberapa kali ALTER TABLE ADD/DROP COLUMN maka mungkin Anda juga sudah tidak hapal lagi sebuah tabel memiliki berapa kolom dan nama serta urutannya bagaimana saja.

Contoh program PHP untuk melakukan query ke MySQL adalah seperti di bawah ini:

```
<?php

#
# contoh-query.php
#
```

```
$host = "localhost";
$user = "..."; # masukkan nama user database yang sebenarnya di sini
$pass = "..."; # masukkan password database yang sebenarnya di sini
$db   = "test"; # masukkan nama database yang sebenarnya di sini

# pertama, konek. nama variabel $conn di sini singkatan dari
#   "connection" (tentu saja anda bebas memakai nama lain)
$conn = mysql_connect($host, $user, $pass) or
  die("Gagal konek!");

# kedua, pilih DB. argumen kedua $conn umumnya tidak perlu
disebutkan
#   lagi kecuali jika ada lebih dari satu koneksi yang anda
lakukan
mysql_select_db($db, $conn) or
  die("Gagal memilih database $db: ".mysql_error());

# ketiga, lakukan query. nama variabel $res di sini singkatan dari
#  "result" (tentu saja anda bebas memakai nama lain). argumen kedua
#  $conn umumnya tidak perlu disebutkan lagi kecuali jika ada lebih
#  dari satu koneksi yang aktif
$res = mysql_query("SELECT * FROM orang", $conn) or
  die("Gagal query: ".mysql_error($conn));

# keempat, ambil baris-baris hasil
$rows = array();
while ($row = mysql_fetch_assoc($res)) $rows[] = $row;

# terakhir, tampilkan (atau proses hasil, dan lain sebagainya)
print_r($rows);

?>
```

`$host` dapat berupa hostname atau IP, yang dapat diikuti nomor port ":port" jika nomor port tidak default (default MySQL adalah 3306). `$host` juga dapat berupa ":path" untuk menyatakan koneksi melalui soket Unix dan bukan soket TCP. path di sini adalah lokasi soket Unix, misalnya `/tmp/mysql.sock` atau `/home/steven/tmp/mysql.sock`.

Hasil program di atas adalah sebagai berikut:

```
Array
(
    [0] => Array
        (
            [id] => 1
            [nama_depan] => Andi
            [nama_belakang] => Firmansyah
            [tanggal_lahir] => 1988-10-01
        )

    [1] => Array
        (
            [id] => 2
            [nama_depan] => Rieke
            [nama_belakang] => Ardianti
            [tanggal_lahir] => 1968-12-28
        )

    [2] => Array
        (
            [id] => 3
            [nama_depan] => Tukiman
            [nama_belakang] =>
            [tanggal_lahir] => 1981-11-01
        )

    [3] => Array
        (
            [id] => 4
            [nama_depan] => Budiman
            [nama_belakang] => Ali
            [tanggal_lahir] => 1981-04-03
        )

    [4] => Array
        (
            [id] => 5
            [nama_depan] => Andi
            [nama_belakang] => Simanjuntak
            [tanggal_lahir] => 1975-06-23
        )
```

```
)
```

Jika ingin menampilkan dalam bentuk tabel (dengan warna berselang-seling misalnya) Anda tinggal ganti baris `print_r()` dengan:

```
echo "<table cellpadding=5 border=1>";
echo "<tr bgcolor=#cccccc>
  <th>ID</th>
  <th>Nama depan</th>
  <th>Nama belakang</th>
  <th>Tanggal lahir</th>
</tr>";

$i = 0;
foreach ($rows as $row) {
  $bgcolor = $i++ % 2 ? "#f0f0f0" : "#ffffff";
  echo "<tr bgcolor=$bgcolor>
    <td>$row[id]</td>
    <td>$row[nama_depan]</td>
    <td>$row[nama_belakang]</td>
    <td>$row[tanggal_lahir]</td>
  </tr>";
}

echo "</table>";
```

dan hasilnya seperti pada Gambar 6-1-1.

ID	Nama depan	Nama belakang	Tanggal lahir
1	Andi	Firmansyah	1988-10-01
2	Rieke	Ardianti	1968-12-28
3	Tukiman		1981-11-01
4	Budiman	Ali	1981-04-03
5	Andi	Simanjuntak	1975-06-23

Gambar 6-1-1.

Memilih Kolom Tertentu Saja Dari Tabel

```
-- Resep 6-1-3: Memilih kolom tertentu saja dari tabel
SELECT nama_kolom1[, nama_kolom2[, ...]] FROM nama_tabel;
```

Anda dapat menyebutkan hanya satu atau dua kolom saja, dalam urutan yang Anda sukai, bahkan menyebutkan sebuah kolom yang sama beberapa kali (meskipun ini pada umumnya tidak ada gunanya sama sekali).

Contoh:

```
mysql> SELECT nama_depan FROM orang;
+------------+
| nama_depan |
+------------+
| Andi       |
| Rieke      |
| Tukiman    |
| Budiman    |
| Andi       |
+------------+
5 rows in set (0.07 sec)
```

SELECT sebagai Kalkulator

```
-- Resep 6-1-4: Menggunakan SELECT sebagai kalkulator
SELECT ekspresi;
```

Tentu saja ekspresi di SELECT tidak hanya melulu berisi * atau nama-nama kolom, tapi bisa berupa ekspresi apa saja yang valid di SQL yang melibatkan angka, string, fungsi, dan lain sebagainya; bahkan bisa mencantumkan query lain atau ekspresi yang tidak melibatkan nama-nama kolom sama sekali. Untuk kasus yang terakhir, tidak dibutuhkan klausa FROM sama sekali.

Contoh:

```
mysql> SELECT 1+1;
+-----+
| 1+1 |
+-----+
|   2 |
```

```
+-----+
1 row in set (0.03 sec)

mysql> SELECT SIN(RADIANS(30));
+-----------------+
| SIN(RADIANS(30)) |
+-----------------+
|         0.500000 |
+-----------------+
1 row in set (0.00 sec)
```

Bisa dilihat bahwa SELECT dapat digunakan sebagai kalkulator di sini. Sayangnya, standar SQL mewajibkan klausa FROM jadi sintaks tanpa FROM ini banyak tidak didukung oleh database lain (tapi setidaknya MySQL dan PostgreSQL membolehkan sintaks ini). Di MySQL 4.1, Anda dapat menggunakan tabel bernama DUAL yang merupakan tabel "dummy" (bohong-bohongan), hanya untuk memuaskan agar klausa FROM ada. Tentu saja Anda juga bisa menggunakan sembarang tabel betulan yang Anda punya. Catatan: DUAL merupakan kata kunci di MySQL.

```
-- Resep 6-1-5: Menggunakan SELECT sebagai kalkulator (dengan tabel
--   dummy)
-- Catatan: MySQL 4.1 ke atas
SELECT ekspresi FROM DUAL;
```

Alias Nama Kolom

```
-- Resep 6-1-6: Alias nama kolom
SELECT ekspresi AS nama_alias_kolom[, ...] FROM nama_tabel;
```

Setiap ekspresi di SELECT dapat Anda beri atau ganti namanya. Ini berguna agar Anda mendapatkan nama sesuai yang diinginkan di bahasa pemrograman, serta berguna juga dalam view dan nested query (akan dijelaskan di bab berikutnya).

Contoh:

```
mysql> SELECT id, id*id FROM orang;
+----+-------+
| id | id*id |
+----+-------+
|  1 |     1 |
```

```
|  2 |     4 |
|  3 |     9 |
|  4 |    16 |
|  5 |    25 |
+----+-------+
5 rows in set (0.02 sec)
```

Di PHP, `mysql_fetch_assoc()` akan menghasilkan array dengan key `id` dan `id*id`. Kadang Anda ingin mengganti id*id ini dengan nama lain, terutama jika ekspresi sudah kompleks. Maka Anda dapat menggunakan:

```
mysql> SELECT id, id*id AS kuadrat FROM orang;
+----+---------+
| id | kuadrat |
+----+---------+
|  1 |       1 |
|  2 |       4 |
|  3 |       9 |
|  4 |      16 |
|  5 |      25 |
+----+---------+
5 rows in set (0.02 sec)
```

Maka di PHP Anda akan mendapati array dengan key `id` dan `kuadrat`.
Catatan: Kata kunci AS bisa dihilangkan di rata-rata produk database, meskipun standar SQL sebetulnya mewajibkan hal ini.

Sebuah contoh lain:

```
mysql> SELECT CONCAT(nama_depan,' ',nama_belakang) AS nama_lengkap
       FROM orang;
+-----------------+
| nama_lengkap    |
+-----------------+
| Andi Firmansyah |
| Rieke Ardianti  |
| Tukiman         |
| Budiman Ali     |
| Andi Simanjuntak |
+-----------------+
5 rows in set (0.01 sec)
```

atau yang lebih afdol lagi (untuk mencegah kita menambahkan spasi di akhir nama depan jika si orang tidak memiliki nama belakang):

```
mysql> SELECT CONCAT(
         '>', – pengapit, hanya untuk demonstrasi keberadaan
spasi
         CASE nama_belakang
           WHEN '' THEN nama_depan
           ELSE CONCAT(nama_depan,' ',nama_belakang)
         END,
         '<' – pengapit, hanya untuk demonstrasi keberadaan spasi
       ) AS nama_lengkap
       FROM orang;
+-------------------+
| nama_lengkap      |
+-------------------+
| >Andi Firmansyah<  |
| >Rieke Ardianti<   |
| >Tukiman<          |
| >Budiman Ali<      |
| >Andi Simanjuntak< |
+-------------------+
5 rows in set (0.01 sec)
```

Catatan: Sayangnya, SQL saat ini tidak mengizinkan kita menyebutkan alias kolom dalam ekspresi di kolom lain. Andaikan bisa, maka kita bisa menulis query dengan ekspresi kompleks dengan lebih singkat lagi. Contoh:

```
-- ini contoh yang tidak valid karena SQL tidak mendukung alias
kolom
--   disebutkan dalam ekspresi lain di SELECT. SQL akan menganggap
--   d sebagai nama kolom biasa dan jika tidak ada, maka query
gagal.
SELECT
  POW(a*a + b*b, 0.5) AS d, – hitung sisi miring siku-siku segitiga
  IF(d = c, "siku-siku",
            IF(d > c, "lancip", "tumpul")
    )
FROM segitiga;
```

```
-- di SQL harus ditulis sebagai berikut:
SELECT
  POW(a*a + b*b, 0.5) AS d, – hitung sisi miring siku-siku segitiga
  IF(d = POW(a*a + b*b, 0.5), “siku-siku”,
           IF(POW(a*a + b*b, 0.5) > c, “lancip”, “tumpul”)
    )
FROM segitiga;
```

Kita akan mempelajari tentang CASE dan IF() sesaat lagi.

Alias Nama Tabel

```
-- Resep 6-1-7: Alias nama tabel
SELECT ... FROM nama_tabel AS nama_alias;
```

Sama seperti nama kolom, nama tabel juga dapat dialiasi. Untuk apa? Ini berguna nanti untuk mempersingkat nama tabel di query multitabel dan di nested query.

Sama pula seperti di alias kolom, kata kunci AS di berbagai database bersifat opsional.

Qualifier Nama Kolom dan Nama Tabel

```
-- Resep 6-1-8: Qualifier nama kolom dan nama tabel
SELECT nama_alias_tabel.nama_kolom ...;

SELECT nama_database.nama_alias_tabel.nama_kolom ...;

SELECT ... FROM nama_database.nama_tabel;
```

Karena, seperti yang nanti akan kita lihat, sebuah perintah SELECT dapat mengambil data dari beberapa tabel sekaligus bahkan tabel yang ada di database yang berbeda, di mana dua buah tabel mungkin saja memiliki kolom dengan nama yang sama, maka di ekspresi SELECT nama kolom dapat kita tambahkan qualifier berupa nama tabel dan nama database.

Catatan: Semua database yang disebutkan dalam sebuah perintah SELECT haruslah terdapat pada server database yang sama. MySQL saat ini tidak mendukung SELECT dari dua database remote yang terpisah satu sama lain.

Catatan: Sintaks a.b untuk nama tabel di database lain seperti Oracle, DB2, dan PostgreSQL sebetulnya berarti qualifier nama tabel dengan nama schema, bukan dengan nama database. Schema (tidak untuk dibingungkan dengan istilah "skema database") merupakan konsep namespace bagi tabel. Kita dapat membuat beberapa tabel dan mengelompokkannya dalam schema terpisah dan tiap schema dapat diberi permission berbeda dari schema lain. Schema termasuk dalam standar SQL, namun MySQL dan beberapa database open source lain saat ini tidak mendukung schema sama sekali.

6.2 Memfilter Baris

Dalam kehidupan nyata, rata-rata tabel database tentu tidak hanya berisi 5 baris melainkan ribuan bahkan hingga jutaan baris. Salah satu fungsi utama query adalah mencari [sedikit] baris [dari sekian banyak baris yang ada] yang memenuhi kriteria tertentu.

Klausa WHERE adalah cara utama yang digunakan untuk memfilter baris. Klausa WHERE menerima ekspresi logika yang akan dievaluasi untuk tiap baris dan jika hasil akhirnya true maka menandakan baris tersebut masuk dalam baris hasil. MySQL juga menerima ekspresi bertipe angka/teks dan jika hasilnya tidak nol maka sama artinya dengan true.

Beberapa contoh:

```
-- 1. sama artinya dengan tanpa WHERE, karena setiap baris akan
--    memiliki hasil evaluasi true. hasilnya 5 baris.
SELECT * FROM orang WHERE true;

-- 2. sama artinya dengan WHERE true
SELECT * FROM orang WHERE 1;

-- 3. hanya pilih orang yang nama depannya Andi. hasilnya 2 baris.
SELECT * FROM orang WHERE nama_depan = 'Andi';

-- 4. hanya pilih orang yang nama depannya Andi dan nama
belakangnya
--    bukan Budiman. hasilnya 1 baris.
SELECT * FROM orang WHERE nama_depan = 'Andi' AND
                          nama_belakang <> 'Firmansyah';

-- 5. kebalikan dari #4. hasilnya 4 baris.
```

```
SELECT * FROM orang WHERE NOT(nama_depan = 'Andi' AND
                              nama_belakang <> 'Budiman');

-- 6. error, karena kolom hobi tidak ada pada tabel orang
SELECT * FROM orang WHERE hobi = 'memancing';

-- 7. hasilnya 0 baris, karena tidak ada yang lahir sebelum 1960
SELECT * FROM orang WHERE tanggal_lahir < '1960-01-01';
```

Dari contoh-contoh di atas bisa dilihat bahwa segala macam ekspresi yang ujung-ujungnya menghasilkan true atau false dapat kita gunakan. Berikut ini akan dibahas elemen-elemen yang membentuk ekspresi logika.

Operator Perbandingan

Sama seperti di rata-rata bahasa pemrograman, dikenal operator perbandingan seperti = (sama dengan), < (lebih kecil), > (lebih besar), != (tidak sama dengan), <=, >=. Untuk lambang tidak sama dengan dapat pula digunakan <>. Operator ini dapat digunakan baik untuk angka, tanggal, maupun string. Di MySQL akan dilakukan konversi seperlunya jika kedua operand yang berbeda tipe dibandingkan (sementara di produk lain seperti PostgreSQL hal ini dilarang, melainkan Anda harus melakukan konversi sendiri dulu secara manual menggunakan `CAST()`).

Perhatikan bahwa operator kondisi untuk mengecek kesamaan di SQL adalah = (tanda sama dengan tunggal), bukan == (tanda sama dengan ganda) seperti di beberapa bahasa pemrograman.

Selain operator-operator perbandingan di atas, SQL juga menawarkan BETWEEN dan IN. BETWEEN untuk mengetes apakah sebuah ekspresi bernilai di antara rentang dua nilai yang disebutkan. Sementara IN untuk mengetes apakah sebuah nilai terdapat dalam sebuah deret/daftar nilai.

```
-- Resep 6-2-1: Sintaks BETWEEN
ekspresi BETWEEN ekpresi1 AND ekspresi2
ekspresi BETWEEN ekpresi1 AND ekspresi2
-- Resep 6-2-2: Sintaks IN
ekspresi IN (ekspresi1[, ekspresi2[, ...]])
ekspresi NOT IN (ekspresi1[, ekspresi2[, ...]])
```

"*a* BETWEEN *b* AND *c*" sebetulnya ekivalen dengan "*a* >= *b* AND *a* <= *c*". Namun BETWEEN terlihat lebih mudah dibaca dan memiliki keuntungan jika *a* berupa ekspresi maka tidak harus dievaluasi dua kali. Contohnya:

```
RAND() BETWEEN 0 AND 0.5
```

ini tidak bisa diganti dengan:

```
RAND() >= 0 AND RAND() <= 0.5
```

karena RAND() akan dievaluasi dua kali dan menghasilkan nilai yang berbeda.

Demikian juga "*a* IN (*b*, *c*, *d*)" sebetulnya dapat diganti dengan "*a* = *b* OR *a* = *c* OR *a* = *d*" namun versi IN lebih ringkas dan tidak perlu mengevaluasi *a* berulang kali. Sebagai tambahan, IN juga dapat menerima query SELECT (subquery) di dalam daftar pilihannya (akan dibahas di Bab 7) dan memiliki potensi untuk dioptimasi oleh optimizer query milik database untuk jumlah pilihan yang banyak.

Operator Logika

Juga sama seperti di rata-rata bahasa pemrograman, dikenal operator boolean AND, OR, dan NOT. MySQL memperkenalkan juga XOR (tidak ada di standar SQL) dan menggunakan lambang-lambang seperti `||` untuk OR dan && untuk AND. Padahal di standar SQL, `||` digunakan untuk menyambung string (di MySQL digunakan fungsi `CONCAT()` untuk menyambung string). Selain itu MySQL juga menggunakan simbol `&`, `|`, `!`, `^` untuk operasi bitwise AND, OR, NOT, XOR.

Untuk menjaga portabilitas query Anda, sebaiknya gunakan saja AND, OR, NOT dan bukan simbol && atau ||. Hindari XOR atau operasi bitwise kecuali benar-benar perlu.

Operator Kondisi (CASE) dan IF()

Operator kondisi CASE dikenal di standar SQL. Gunanya sama seperti di bahasa pemrograman yaitu untuk memilih satu dari beberapa alternatif yang ada. Sintaksnya:

```
-- Resep 6-2-3: Sintaks CASE
CASE ekspresi
   WHEN nilai1 THEN hasil1
  [WHEN nilai2 THEN hasil2 [...]]
  [ELSE hasilN]
END
```

Ekspresi, *nilaiX*, dan *hasilX* semua adalah ekspresi. Jika ekspresi bernilai *nilai1*, maka hasil dari CASE adalah *hasil1*. Jika *nilai2*, maka hasilnya *hasil2*, dan seterusnya. Jika tidak ada yang cocok, dan terdapat klausa ELSE maka *hasilN* yang dipilih. Jika tidak ada yang cocok tapi tidak ada klausa ELSE, maka hasil dari CASE adalah NULL.

Beberapa contoh CASE:

```
-- mengkonversi kode bulan 3 huruf menjadi nama penuhnya
CASE kode_bulan
  WHEN 'Jan' THEN 'Januari'
  WHEN 'Feb' THEN 'Februari'
    WHEN 'Peb' THEN 'Februari' – alternatif ejaan untuk Feb
  WHEN 'Mar' THEN 'Maret'
  WHEN 'Apr' THEN 'April'
  WHEN 'Mei' THEN 'Mei'
  WHEN 'Jun' THEN 'Juni'
  WHEN 'Jul' THEN 'Juli'
  WHEN 'Agu' THEN 'Agustus'
    WHEN 'Agt' THEN 'Agustus' – alternatif untuk 'Agu'
  WHEN 'Sep' THEN 'September'
  WHEN 'Okt' THEN 'Oktober'
  WHEN 'Nop' THEN 'Nopember'
    WHEN 'Nov' THEN 'Nopember' - alternatif untuk 'Nop'
  WHEN 'Des' THEN 'Desember'
END

-- menentukan besar diskon. ini contoh CASE bersarang (nested CASE,
--   CASE dalam CASE)
CASE jumlah_beli
  WHEN 1 THEN 0
  WHEN 2 THEN
    CASE barang
      WHEN 'fx 800' THEN 0.075
      WHEN 'fx 850' THEN 0.075
      WHEN 'fx 900' THEN 0.075
      ELSE 0
    END
  ELSE 0.1
END
```

Contoh terakhir kurang lebih berbunyi, "Kalau beli hanya 1 barang, tidak ada diskon. Kalau beli 2 barang, maka lihat dulu barangnya apa. Kalau salah satu dari 'fx 800', 'fx 850', atau 'fx 950' maka berikan diskon 7,5%. Kalau bukan barang itu, tidak ada diskon. Kalau beri barang lebih dari 2, maka beri diskon 10%."

Catatan untuk MySQL: Sebagai tambahan dari CASE, MySQL juga memiliki fungsi

`IF()` yang kegunaannya mirip:

```
-- Resep 6-2-4: Sintaks fungsi IF()
IF(ekspresi, hasil1, hasil2)
```

Jika *ekspresi* menghasilkan nilai benar, maka *hasil1* yang dipilih. Jika salah, *hasil2*. Fungsi IF() ini mirip dengan yang sering kita jumpai di spreadsheet seperti Lotus atau Excel dan bentuknya memang lebih ringkas daripada CASE untuk kasus memilih 2 alternatif. Untuk memilih 3 atau lebih alternatif, bisa digunakan `IF()` yang bersarang (`IF()` dalam `IF()`). Sebagai contoh, kode CASE untuk memilih diskon di atas dapat ditulis di MySQL dalam bentuk IF() sbb:

```
IF(jumlah_beli = 1,
   0,
   IF(jumlah_beli = 2,
      IF(produk = 'fx 800' OR
         produk = 'fx 850' OR
         produk = 'fx 900', 0.075, 0
      ),
      0.1
   )
)
```

`IF()` bukan merupakan bagian dari standar SQL, jadi sebaiknya dihindari dan gunakan saja CASE.

Operator IS NULL, IS NOT NULL

Untuk membandingkan apakah sebuah kolom atau ekspresi bernilai NULL atau bukan, tidak dapat digunakan operator kondisi `=` atau `!=` karena, ingat pembahasan di Bab 3, NULL tidak *sama* dengan nilai apapun juga termasuk NULL itu sendiri. Sebagai alternatifnya, SQL menyediakan operator IS NULL dan IS NOT NULL.

```
-- Resep 6-2-5: Mengecek nilai NULL atau bukan
ekspresi IS NULL – 1. mengecek apakah ekspresi NULL
ekspresi IS NOT NULL – 2. mengecek apakah ekspresi bukan NULL
NOT (ekspresi IS NULL) – idem #2
```

Kolom di Klausa WHERE

Kolom di klausa WHERE adalah nama-nama kolom yang terdapat pada relasi yang disebutkan di klausa FROM. Namun jika ada alias kolom yang disebutkan, maka alias tersebut yang dipakai. Contoh:

```
SELECT o.nama_depan depan FROM orang o WHERE depan='Andi';
```

Di sini, kita bisa juga menuliskan *orang.depan* atau *orang_depan*, namun jika seandainya tabel *orang* mengandung juga nama kolom bernama *depan*, maka akan "tertutupi" oleh alias kolom *depan*.

Singkatnya, WHERE menyadari keberadaan alias yang disebutkan di ekspresi SELECT.

Menghilangkan Baris-Baris Dobel

```
-- Resep 6-2-6: Menghilangkan baris-baris dobel
SELECT DISTINCT ... FROM ...;
mysql> SELECT nama_depan FROM orang;
+------------+
| nama_depan |
+------------+
| Andi       |
| Rieke      |
| Tukiman    |
| Budiman    |
| Andi       |
+------------+
5 rows in set (0.00 sec)

mysql> SELECT DISTINCT nama_depan FROM orang;
+------------+
| nama_depan |
+------------+
| Andi       |
| Rieke      |
| Tukiman    |
| Budiman    |
+------------+
4 rows in set (0.09 sec)
```

DISTINCT berguna untuk menghilangkan baris yang dobel. Tanpa opsi DISTINCT, kita menjumpai ada 2 orang yang memiliki nama depan Andi. Dengan DISTINCT, baris yang dobel ini dihilangkan.

Catat bahwa seandainya kita memberikan perintah

```
SELECT DISTINCT id, nama_depan FROM orang;
```

atau

```
SELECT DISTINCT nama_depan, nama_belakang FROM orang;
```

maka hasilnya tetap 5 baris dikarenakan tidak ada baris yang dobel. Meskipun *nama_depan* ada yang bernilai sama, namun kombinasi (*id*, *nama_depan*) maupun (*nama_depan*, *nama_belakang*) bersifat unik untuk tabel tersebut.

6.3 Mengurutkan Baris Hasil

Menyortir juga sama pentingnya dengan memfilter dalam query. Dari sekian banyak data yang ada, seringkali kita ingin melihat sebagian data mulai dari yang "*paling* ..." atau yang "*paling tidak* ..." (mis: paling muda, paling tua, paling panjang namanya, paling besar saldonya, paling sering memesan barang, dan lain sebagainya).

Untuk melakukan penyortiran di SQL, digunakan klausa ORDER BY.

```
-- Resep 6-3-1: Sintaks dasar klausa ORDER BY

SELECT ... [FROM ... [WHERE ...]]
  ORDER BY    ekspresi1 [ASC | DESC],
            [, ekspresi2 [ASC | DESC]
            [, ...];
```

Sama seperti pada klausa WHERE, ORDER BY dapat mengurutkan berdasarkan sembarang ekspresi, tidak hanya nama kolom saja. Dan sama seperti klausa WHERE, ORDER BY juga menyadari keberadaan alias nama kolom/alias nama tabel.

ORDER BY harus selalu disebutkan di belakang klausa WHERE, jika keduanya ada. *Ekspresi1* adalah kriteria pertama untuk penyortiran. Jika ada dua nilai yang sama, barulah digunakan *ekspresi2* sebagai kriteria kedua (jika memang disebutkan pada query). ASC dan DESC artinya urutan penyortiran; ASC berarti menaik dari kecil ke besar sementara DESC menurun dari besar ke kecil. Jika tidak disebutkan, defaultnya adalah DESC.

Contoh:

```
-- mengurutkan berdasarkan nama depan
mysql> SELECT * FROM orang ORDER BY nama_depan;
+----+-----------+--------------+--------------+
| id | nama_depan | nama_belakang | tanggal_lahir |
+----+-----------+--------------+--------------+
|  1 | Andi       | Firmansyah    | 1988-10-01    |
|  5 | Andi       | Simanjuntak   | 1975-06-23    |
|  4 | Budiman    | Ali           | 1981-04-03    |
|  2 | Rieke      | Ardianti      | 1968-12-28    |
|  3 | Tukiman    |               | 1981-11-01    |
+----+-----------+--------------+--------------+
5 rows in set (0.00 sec)

-- mengurutkan berdasarkan nama depan juga, tapi terbalik
mysql> SELECT * FROM orang ORDER BY nama_depan DESC;
+----+-----------+--------------+--------------+
| id | nama_depan | nama_belakang | tanggal_lahir |
+----+-----------+--------------+--------------+
|  3 | Tukiman    |               | 1981-11-01    |
|  2 | Rieke      | Ardianti      | 1968-12-28    |
|  4 | Budiman    | Ali           | 1981-04-03    |
|  1 | Andi       | Firmansyah    | 1988-10-01    |
|  5 | Andi       | Simanjuntak   | 1975-06-23    |
+----+-----------+--------------+--------------+
5 rows in set (0.01 sec)

-- mengurutkan berdasarkan fungsi, CHAR_LENGTH() mengembalikan
panjang
--   sebuah string (jumlah karakter yang ada pada string). plus
--   kombinasi dengan klausa WHERE
mysql> SELECT nama_belakang FROM orang
       WHERE nama_belakang <> ''
       ORDER BY CHAR_LENGTH(nama_belakang);
+--------------+
| nama_belakang |
+--------------+
| Ali           |
| Ardianti      |
| Firmansyah    |
| Simanjuntak   |
+--------------+
```

```
4 rows in set (0.00 sec)

-- sama seperti sebelumnya, tapi kita menambahkan ekspresi yang
ingin
--   diurutkan ke dalam ekspresi SELECT dan menyebutkan aliasnya di
--  klausa ORDER BY
mysql> SELECT
         nama_belakang,
         CHAR_LENGTH(nama_belakang) pjg
       FROM orang
       WHERE nama_belakang <> ''
       ORDER BY pjg;
+--------------+------+
| nama_belakang | pjg  |
+--------------+------+
| Ali           |    3 |
| Ardianti      |    8 |
| Firmansyah    |   10 |
| Simanjuntak   |   11 |
+--------------+------+
4 rows in set (0.03 sec)

-- mengurutkan berdasarkan nama depan, jika ada yang sama maka
urutkan
--   berdasarkan tanggal lahir (dari yang paling tua)
mysql> SELECT * FROM orang ORDER BY nama_depan, tanggal_lahir;
+----+------------+--------------+--------------+
| id | nama_depan | nama_belakang | tanggal_lahir |
+----+------------+--------------+--------------+
|  5 | Andi       | Simanjuntak   | 1975-06-23    |
|  1 | Andi       | Firmansyah    | 1988-10-01    |
|  4 | Budiman    | Ali           | 1981-04-03    |
|  2 | Rieke      | Ardianti      | 1968-12-28    |
|  3 | Tukiman    |               | 1981-11-01    |
+----+------------+--------------+--------------+
5 rows in set (0.02 sec)
```

Karena ORDER BY dapat diisi ekspresi sembarang, bahkan yang tidak menyinggung salah satu kolom yang ada sama sekali, maka kita bisa melakukan hal seperti ini:

```
-- Resep 6-3-2: "Mengacak" tabel, menampilkan baris dalam urutan
acak
SELECT ... FROM ... ORDER BY RAND();
```

Contoh:

```
mysql> SELECT * FROM orang ORDER BY RAND ();
+----+-----------+--------------+--------------+
| id | nama_depan | nama_belakang | tanggal_lahir |
+----+-----------+--------------+--------------+
|  5 | Andi       | Simanjuntak   | 1975-06-23    |
|  2 | Rieke      | Ardianti      | 1968-12-28    |
|  1 | Andi       | Firmansyah    | 1988-10-01    |
|  4 | Budiman    | Ali           | 1981-04-03    |
|  3 | Tukiman    |               | 1981-11-01    |
+----+-----------+--------------+--------------+
5 rows in set (0.00 sec)

– hasilnya akan berbeda setiap kali
mysql> SELECT * FROM orang ORDER BY RAND();
+----+-----------+--------------+--------------+
| id | nama_depan | nama_belakang | tanggal_lahir |
+----+-----------+--------------+--------------+
|  4 | Budiman    | Ali           | 1981-04-03    |
|  3 | Tukiman    |               | 1981-11-01    |
|  2 | Rieke      | Ardianti      | 1968-12-28    |
|  1 | Andi       | Firmansyah    | 1988-10-01    |
|  5 | Andi       | Simanjuntak   | 1975-06-23    |
+----+-----------+--------------+--------------+
5 rows in set (0.00 sec)
```

Fungsi RAND() akan menghasilkan angka pecahan acak antara 0 dan 1.

ORDER BY dan NULL

Jika kolom yang ingin Anda sortir mengandung NULL, Anda perlu menentukan ingin urutan yang seperti bagaimana. MySQL menaruh NULL dalam urutan teratas (kecuali jika menyebutkan DESC, di mana urutan akan dibalik sehingga NULL di akhir). Database lain seperti PostgreSQL dan Firebird memiliki kelakuan sebaliknya, yaitu menaruh NULL di akhir untuk ASC dan di awal untuk DESC; kelakuan ini sesuai dengan standar SQL. MySQL bisa saja membalik urutan ini di kemudian hari, atau bisa juga Anda ingin NULL selalu di awal atau di akhir tak

peduli arah pengurutan ASC atau DESC.

Untuk membuat pengurutan NULL lebih pasti dan portabel, bisa digunakan resep berikut:

```
-- Resep 6-3-3: Selalu menaruh NULL di awal
SELECT ... ORDER BY nama_kolom IS NULL DESC, nama_kolom
[ASC|DESC], ...;

-- Resep 6-3-4: Selalu menaruh NULL di akhir
SELECT ... ORDER BY nama_kolom IS NULL, nama_kolom [ASC|DESC], ...;
```

Resep ini bekerja dengan cara menambahkan kategori pertama berupa ekspresi (*nama_kolom* IS NULL), baru kategori pengurutan sebenarnya yang diinginkan, *nama_kolom*. Jika kolom bernilai NULL maka hasilnya adalah true. True lebih besar daripada false. Jadi pada resep 6-3-3 kita selalu menaruh NULL di atas karena kita menaruh true (yang lebih besar dari false) di atas dengan DESC. Sebaliknya di 6-3-4 kita menaruh true di bawah sehingga NULL selalu di akhir. Tentu saja, ekspresi di resep 6-3-3 bisa juga diganti dengan ekspresi (*nama_kolom* IS NOT NULL).

6.4 MAX(), MIN(), COUNT(), SUM(), AVG()

Sejauh ini kita membahas query SQL yang dapat mengembalikan baris-baris dari tabel dengan kemampuan memfilter dan mengurutkan. Namun salah satu kemampuan lain SQL yang amat berguna dalam reporting adalah meringkas data (summarizing), termasuk di sini yaitu mentotal (menggunakan fungsi SUM()), menghitung rata-rata (AVG()), mencari nilai maksimum dan minimum (MAX(), MIN()), menghitung jumlah baris (COUNT()), dan lain sebagainya.

Fungsi-fungsi ini disebut group function. Group function berbeda dengan fungsi biasa karena kalau fungsi biasa mengembalikan satu baris hasil untuk satu baris masukan, maka group function akan mengembalikan satu baris hasil saja untuk semua baris masukan. Contohnya:

```
-- menghitung ada berapa baris di tabel orang
mysql> SELECT COUNT(nama_depan) FROM orang;
+------------------+
| count(nama_depan) |
+------------------+
```

```
|                5 |
+------------------+
1 row in set (0.05 sec)

-- menghitung ada berapa baris di tabel orang yang nama_depan-nya
Andi
mysql> SELECT COUNT(nama) FROM orang WHERE nama_depan='Andi';
+------------------+
| count(nama_depan) |
+------------------+
|                2 |
+------------------+
1 row in set (0.03 sec)
```

Hasil query ini hanya satu baris dikarenakan adanya group function COUNT(), tak peduli berapapun jumlah baris yang ada pada tabel *orang*.

Beberapa contoh lain:

```
CREATE TABLE angka (i INT);
INSERT INTO angka VALUES (1);
INSERT INTO angka VALUES (2);
INSERT INTO angka VALUES (3);
INSERT INTO angka VALUES (7);
INSERT INTO angka VALUES (4);
INSERT INTO angka VALUES (6);
INSERT INTO angka VALUES (6);
INSERT INTO angka VALUES (10);
INSERT INTO angka VALUES (0);
INSERT INTO angka VALUES (7);

— menghitung jumlah baris di tabel angka
mysql> SELECT COUNT(i) FROM angka;
+----------+
| COUNT(i) |
+----------+
|       10 |
+----------+
1 row in set (0.00 sec)

-- menghitung jumlah angka yang genap
```

```
mysql> SELECT COUNT(i) FROM angka WHERE i % 2 = 0;
+----------+
| COUNT(i) |
+----------+
|        6 |
+----------+
1 row in set (0.00 sec)

– berapa total jumlah semua angka?
mysql> SELECT SUM(i) FROM angka;
+--------+
| SUM(i) |
+--------+
|     46 |
+--------+
1 row in set (0.01 sec)

– berapa total jumlah angka yang ganjil?
mysql> SELECT SUM(i) FROM angka WHERE i % 2 = 1;
+--------+
| SUM(i) |
+--------+
|     18 |
+--------+
1 row in set (0.00 sec)

– apa angka yang terkecil dan terbesar?
mysql> SELECT MIN(i), MAX(i) FROM angka;
+--------+--------+
| MIN(i) | MAX(i) |
+--------+--------+
|      0 |     10 |
+--------+--------+
1 row in set (0.00 sec)

-- rata-ratakan angka
+--------+
| AVG(i) |
+--------+
| 4.6000 |
+--------+
```

```
1 row in set (0.00 sec)

-- rata-ratakan angka, tapi jangan masukkan dalam perhitungan
jika nol
--  bisa dilihat bahwa nilai rata-rata akan meningkat
mysql> SELECT AVG(i) FROM angka WHERE i <> 0;
+--------+
| AVG(i) |
+--------+
| 5.1111 |
+--------+
1 row in set (0.01 sec)

-- ekivalen dengan AVG(), tapi tidak mengecek kasus di mana
COUNT()
--   sama dengan 0 (tidak ada yang dirata-ratakan)
SELECT SUM(i)/COUNT(i) FROM angka;
```

Selain keempat fungsi ini, masih ada beberapa group function lain seperti STD()/ STDDEV() dan VARIANCE() untuk menghitung simpangan baku dan varians (statistik) dan GROUP_CONCAT() yang diperkenalkan di MySQL 4.1 yang berkelakuan seperti SUM() tapi alih-alih menjumlahkan angka, fungsi ini menggabungkan string.

Group Function dan Ekspresi

Sebetulnya group function dapat menerima ekspresi sembarang, tidak hanya menerima nama kolom sebagai argumennya. Beberapa contoh:

```
-- ekivalen dengan COUNT()
SELECT SUM(1) FROM angka;

-- menghitung harga total sebuah pesanan (pesanan dengan id 10), di
--   mana harga total dihitung dengan mengalikan jumlah tiap barang
--   dengan harga satuan barang
SELECT SUM(jumlah * harga_satuan) AS harga_total
FROM detail_pesanan
WHERE id_pesanan=10;

-- menggabungkan SUM dengan CASE, untuk menghitung jumlah baris yang
--   memenuhi kriteria tertentu. di sini kita ingin menghitung
jumlah
```

```
--   angka yang genap (yaitu di mana i % 2 sama dengan 0).
SELECT SUM(CASE i % 2 WHEN 0 THEN 1 ELSE 0 END) FROM angka;
```

Contoh terakhir ekivalen dengan COUNT() dengan klausa WHERE, namun perbedaannya kondisi kita taruh di dalam group function. Bentuk ini dapat dipakai untuk melakukan penghitungan jumlah baris yang memenuhi beberapa kriteria tertentu dalam satu buah query saja.

```
-- Resep 6-4-1: Menghitung sekaligus jumlah baris yang memenuhi
--              beberapa kriteria berbeda
SELECT
    SUM(CASE ekspresi1 WHEN memenuhi_kriteria THEN 1 ELSE 0 END)
  [,SUM(CASE ekspresi2 WHEN memenuhi_kriteria THEN 1 ELSE 0 END)]
  [,...]
  FROM ...;
```

Contoh, untuk menghitung jumlah anggota yang pria, wanita, dan tidak diketahui kelaminnya kita bisa menggunakan tiga buah query yang mengandung COUNT() + WHERE.

```
SELECT COUNT(id) FROM anggota WHERE kelamin = 'L'; – jumlah yg pria
SELECT COUNT(id) FROM anggota WHERE kelamin = 'P'; – jumlah yg
wanita
SELECT COUNT(id) FROM anggota WHERE kelamin IS NULL; – yg tdk diket.
```

namun dengan SUM(CASE ...) kita bisa melakukan semuanya dalam satu query saja:

```
SELECT
  SUM(CASE kelamin WHEN 'L' THEN 1 ELSE 0 END) as jumlah_pria,
  SUM(CASE kelamin WHEN 'P' THEN 1 ELSE 0 END) as jumlah_wanita,
  SUM(CASE kelamin IS NULL WHEN TRUE THEN 1 ELSE 0 END) as
jumlah_td
FROM anggota;
```

Group Function dan NULL

Bagaimana efek NULL pada berbagai group function? Mari kita tambahkan satu buah baris berisi nilai NULL pada tabel *angka*.

```
INSERT INTO angka VALUES (NULL);
```

dan melihat efeknya:

```
mysql> SELECT count(i), min(i), max(i), sum(i), avg(i) FROM angka
       WHERE i IS NOT NULL;
+----------+--------+--------+--------+--------+
| count(i) | min(i) | max(i) | sum(i) | avg(i) |
+----------+--------+--------+--------+--------+
|       10 |      0 |     10 |     46 | 4.6000 |
+----------+--------+--------+--------+--------+
1 row in set (0.00 sec)

mysql> SELECT count(i), min(i), max(i), sum(i), avg(i) FROM
angka;
+----------+--------+--------+--------+--------+
| count(i) | min(i) | max(i) | sum(i) | avg(i) |
+----------+--------+--------+--------+--------+
|       10 |      0 |     10 |     46 | 4.6000 |
+----------+--------+--------+--------+--------+
1 row in set (0.00 sec)
```

Ternyata tidak ada bedanya. COUNT(*i*) menghitung jumlah kolom *i* yang tidak NULL. Jadi jika ada baris yang mengandung *i* NULL maka tidak akan ikut dihitung. Bisa juga dikatakan bahwa COUNT(*i*) berarti "berapa buah jumlah *i* yang diketahui?" Demikian pula dengan MIN(), MAX(), SUM(), dan AVG() tidak mengacuhkan NULL.

COUNT(*) dan COUNT(DISTINCT)

Khusus untuk COUNT(), dikenal dua bentuk khusus:

```
-- Resep 6-4-2: Bentuk khusus COUNT()

-- menghitung semua baris, tak peduli apakah ada yang NULL atau
tidak
COUNT(*)

-- menghitung jumlah ekspresi yang unik
COUNT(DISTINCT ekspresi)
```

COUNT(*) akan menghitung jumlah baris, tak peduli apakah ada kolom yang bernilai NULL atau tidak. Sementara COUNT(DISTINCT ...) menghitung jumlah ekspresi yang bernilai unik. Contoh:

```
mysql> SELECT COUNT(i) FROM angka;
```

```
+----------+
| COUNT(i) |
+----------+
|       10 |
+----------+
1 row in set (0.01 sec)

– baris NULL yang kita tambahkan sebelumnya kini ikut dihitung
mysql> SELECT COUNT(*) FROM angka;
+----------+
| COUNT(*) |
+----------+
|       11 |
+----------+
1 row in set (0.01 sec)

– angka 6 dan 7 terdapat di lebih dari satu baris, maka total
–   angka yang unik hanya ada 10-2 = 8 buah
mysql> SELECT COUNT(DISTINCT i) FROM angka;
+------------------+
| COUNT(DISTINCT i) |
+------------------+
|                8 |
+------------------+
1 row in set (0.02 sec)
```

6.5 Pengelompokan Data (Klausa GROUP BY)

Anda mungkin bertanya, kenapa fungsi-fungsi yang dijelaskan di subbab sebelumnya, COUNT(), SUM(), dan lain sebagainya disebut *group function* (dan bukan summary function atau, katakanlah, one-line-result function)? Ini karena fungsi-fungsi ini dapat bekerja sama dengan klausa GROUP BY yang dijelaskan di subbab ini.

```
-- Resep 6-5-1: Sintaks dasar GROUP BY
SELECT ... FROM ...
  [WHERE ...]
  GROUP BY ekspresi[, ekspresi[, ...]]
  [ORDER BY ...];
```

6.5 Pengelompokan Data (Klausa GROUP BY)

GROUP BY berguna untuk meringkas baris-baris menjadi kelompok-kelompok. Berikut ini dua buah contoh untuk ilustrasi:

```
-- tanpa klausa GROUP BY
mysql> SELECT nama_depan FROM orang;
+------------+
| nama_depan |
+------------+
| Andi       |
| Rieke      |
| Tukiman    |
| Budiman    |
| Andi       |
+------------+
5 rows in set (0.02 sec)

– tanpa GROUP BY
mysql> SELECT COUNT(*) FROM orang;
+----------+
| COUNT(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)

-- dengan GROUP BY
mysql> SELECT nama_depan, COUNT(*) FROM orang GROUP BY
nama_depan;
+------------+----------+
| nama_depan | COUNT(*) |
+------------+----------+
| Andi       |        2 |
| Budiman    |        1 |
| Rieke      |        1 |
| Tukiman    |        1 |
+------------+----------+
4 rows in set (0.00 sec)
```

Bisa dilihat bahwa GROUP BY membuat COUNT(*) tidak hanya menghasilkan satu baris, melainkan tiap baris untuk nilai *nama_depan* yang berbeda. Dengan kata lain, untuk setiap group yang dibentuk/didefinisikan oleh klausa GROUP BY.

Tanpa kehadiran GROUP BY, hanya ada satu group yaitu keseluruhan baris.

GROUP BY memang hanya masuk akal digunakan jika bersama-sama dengan group function. GROUP BY berguna untuk melakukan *breakdown*, misalnya untuk mengetahui total pemasukan tiap propinsi, tiap periode, tiap propinsi *dan* periode, dan lain sebagainya.

Klausa GROUP BY selalu disebutkan setelah klausa WHERE (jika ada) dan sebelum ORDER BY (jika ada).

Beberapa contoh lain:

```
-- hitung berapa total nilai tiap pesanan
SELECT SUM(jumlah * harga_satuan)
FROM detail_pesanan
GROUP BY id_pesanan;

-- hitung total pendapatan tiap bulan, urutkan dari yang terbaru
SELECT SUM(earning)
FROM earning
GROUP BY YEAR(date), MONTH(date)
ORDER BY YEAR(date) DESC, MONTH(date) DESC;
```

Contoh yang terakhir mendemonstrasikan pengelompokan berdasarkan ekspresi. Fungsi-fungsi tanggal dan waktu akan dibahas di subbab berikutnya.

Jika kita hanya melakukan pengelompokan BY YEAR(date), maka kita tidak dapat melihat per bulan dalam satu tahun karena semua akan "terlipat" ke dalam 1 baris tahun yang sama. Sementara kalau kita hanya melakukan pengelompokan BY MONTH(date) maka beberapa tahun yang berbeda akan terlipat juga dalam 1 dari 12 bulan yang ada dan hasil akhirnya maksimum hanya 12 baris (sesuai jumlah bulan yang mungkin dalam satu tahun kalender). Karena itu kita melakukan pengelompokan berdasarkan tahun *dan* bulan.

Klausa HAVING

```
-- Resep 6-5-2: Sintaks HAVING
SELECT ... FROM ... [WHERE ...]
  GROUP BY ...
  HAVING ekspresi
  ...;
```

6.5 Pengelompokan Data (Klausa GROUP BY)

Klausa HAVING sama dengan WHERE yaitu untuk menyeleksi baris, bedanya adalah, klausa WHERE dievaluasi terlebih dulu untuk setiap baris masukan (baris dari tabel yang disebutkan) sementara klausa HAVING dievaluasi setelah database melakukan pengelompokan GROUP BY. Karena itu di klausa HAVING diperbolehkan *ekspresi* yang menyebutkan group function sementara di WHERE tidak boleh. Bisa dikatakan bahwa HAVING adalah WHERE-nya GROUP BY.

Klausa HAVING disebutkan setelah GROUP BY. Dan hanya bisa disebutkan jika ada klausa GROUP BY, tentu saja. Beberapa contoh:

```
-- 1. Daftarkan pesanan yang nilai totalnya lebih dari 200rb
rupiah.
--    Ini tidak bisa dilakukan dengan klausa WHERE karena di klausa
--    WHERE nilai total belum terhitung.
SELECT SUM(jumlah * harga_satuan)
FROM detail_pesanan
GROUP BY id_pesanan
HAVING SUM(jumlah * harga_satuan) > 200000;

-- 2. Sama dengan #1, hanya menggunakan alias kolom
SELECT SUM(jumlah * harga_satuan) AS nilai_total
FROM detail_pesanan
GROUP BY id_pesanan
HAVING nilai_total > 200000;

-- 3. Daftarkan nilai total pesanan, tapi tanpa menghitung barang X.
--    Untuk yang ini, penyeleksian dilakukan di klausa WHERE.
SELECT SUM(jumlah * harga_satuan)
FROM detail_pesanan
WHERE produk <> 'X'
GROUP BY id_pesanan;

-- 4. Nama depan mana yang dobel? Menggunakan HAVING, bukan WHERE.
SELECT nama_depan, COUNT(*)
FROM orang
GROUP BY nama_depan
HAVING COUNT(*) > 1;

-- hasil query #4
+------------+----------+
| nama_depan | COUNT(*) |
```

```
+-----------+---------+
| Andi      |       2 |
+-----------+---------+
1 row in set (0.00 sec)
```

ROLLUP

```
-- Resep 6-5-3: Sintaks ROLLUP
SELECT ... GROUP BY ekspresi[, ekspresi[, ...] WITH ROLLUP;
```

GROUP BY di MySQL dan beberapa database lain seperti MS SQL Server memberi opsi WITH ROLLUP. Dengan opsi ini, database akan menambahkan satu baris berisi total. Contoh:

```
mysql> SELECT nama_depan, COUNT(nama_depan)
       FROM orang GROUP BY nama_depan WITH ROLLUP;
+-----------+-----------------+
| nama_depan | COUNT(nama_depan) |
+-----------+-----------------+
| Andi      |               2 |
| Budiman   |               1 |
| Rieke     |               1 |
| Tukiman   |               1 |
| NULL      |               5 |
+-----------+-----------------+
5 rows in set (0.02 sec)
```

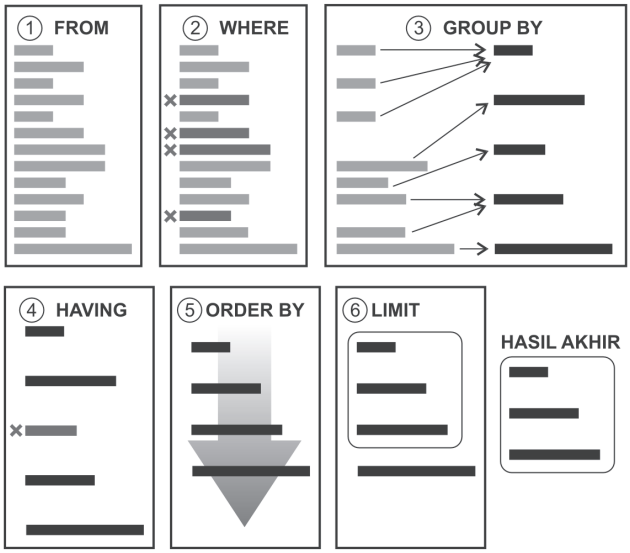
6.6 Membatasi Jumlah Baris (LIMIT)

Kita telah melihat elemen-elemen dasar SELECT, yaitu pemfilteran baris masukan (klausa WHERE), pengelompokan (GROUP BY), pemfilteran baris hasil pengelompokan (HAVING), dan pengurutan hasil (ORDER BY). Database mengerjakan query dalam urutan sesuai urutan penyebutan klausa:

1. Baris-baris masukan dari relasi (tabel) yang disebutkan di FROM diambil.
2. Untuk setiap baris masukan ini, klausa WHERE dievaluasi. Baris yang tidak memenuhi persyaratan dibuang.
3. Ekspresi SELECT dikalkulasi untuk menghasilkan baris keluaran.
4. Jika terdapat group function di ekspresi SELECT, maka kelompokkan baris-baris sesuai definisi GROUP BY.
5. Untuk setiap baris hasil dari #4, terapkan kriteria HAVING (jika ada). Semua

baris yang tidak memenuhi persyaratan HAVING dibuang.

6. Urutkan baris hasil sesuai klausa ORDER BY.
   Ada satu lagi tahap terakhir, yaitu pemotongan (clipping) menggunakan klausa LIMIT.
7. Pilih hanya sekian baris dari #6 menurut klausa LIMIT.
   Jadi secara lengkapnya, pemrosesan sebuah query SQL bisa dilihat seperti pada Gambar 6-6-1.



Gambar 6-6-1

Resep 6-6-1: Sintaks klausa LIMIT

```
SELECT ... LIMIT [offset, ]jumlah;

-- sintaks alternatif, lebih jelas
SELECT ... LIMIT jumlah OFFSET offset;
```

Klausa LIMIT biasa digunakan dalam *paging*, yaitu untuk mengambil sekian kelompok baris hasil ("halaman") pertama saja atau kelompok baris hasil berikutnya ("halaman" kedua), dan seterusnya.

Klausa LIMIT disebutkan di akhir setelah klausa-klausa lainnya (WHERE, ORDER BY, GROUP BY). Jika *offset* tidak disebutkan, defaultnya 0 (artinya ambil *jumlah* baris dari awal). Jika *offset* disebutkan, artinya lewati dulu sejumlah baris menurut *offset* baru ambil *jumlah* buah baris.

Klausa LIMIT sebetulnya hanya masuk akal jika kita sebutkan bersamaan dengan ORDER BY. Kalau kita meminta "Ambil 5 buah item pertama ..." maka tentunya akan timbul pertanyaan "5 item pertama berdasarkan apanya? Namanya? Popularitasnya? Panjangnya? Dan lain sebagainya." Namun MySQL mengizinkan kita tidak menyebutkan ORDER BY, dan urutan baris hasil adalah sesuai dengan kronologi INSERT.

MySQL adalah salah satu database pertama yang memperkenalkan klausa LIMIT. Saking bergunanya dan seringnya paging digunakan, maka beberapa database open source lainnya kini ikut menambahkan klausa serupa walaupun sintaksnya agak berbeda; di PostgreSQL hanya dikenal sintaks LIMIT yang menggunakan OFFSET; di Firebird dikenal sintaks SELECT FIRST ... SKIP .... Bahkan database seperti MS SQL Server pun kini menambahkan SELECT TOP ... (walaupun tanpa mendukung offset). Klausa LIMIT amat praktis sebab sebelumnya untuk melakukan paging dibutuhkan cursor. Cursor lebih repot digunakan karena harus didefinisikan dulu dan diatur lagi dalam perintah SQL yang berbeda, sementara LIMIT menyatu dalam perintah SELECT.

Beberapa contoh query yang menggunakan klausa LIMIT:

```
-- 1. Ambil 2 orang pertama
mysql> SELECT id, nama_depan, nama_belakang FROM orang
       ORDER BY id LIMIT 2;
+----+------------+--------------+--------------+
| id | nama_depan | nama_belakang | tanggal_lahir |
+----+------------+--------------+--------------+
|  1 | Andi       | Firmansyah    | 1988-10-01    |
|  2 | Rieke      | Ardianti      | 1968-12-28    |
+----+------------+--------------+--------------+
2 rows in set (0.07 sec)

-- 2. Ambil 2 orang berikutnya setelah #1
mysql> SELECT id, nama_depan, nama_belakang FROM orang
       ORDER BY id LIMIT 2 OFFSET 2;
+----+------------+--------------+--------------+
| id | nama_depan | nama_belakang | tanggal_lahir |
+----+------------+--------------+--------------+
|  3 | Tukiman    |              | 1981-11-01    |
|  4 | Budiman    | Ali          | 1981-04-03    |
+----+------------+--------------+--------------+
2 rows in set (0.01 sec)
```

```
-- 3. Ambil 2 orang berikutnya lagi; karena tabel orang berisi 5
baris
--    saja, maka query kali ini hanya menghasilkan 1 baris
mysql> SELECT id, nama_depan, nama_belakang FROM orang
       ORDER BY id LIMIT 2 OFFSET 4;
+----+------------+--------------+--------------+
| id | nama_depan | nama_belakang | tanggal_lahir |
+----+------------+--------------+--------------+
|  5 | Andi       | Simanjuntak   | 1975-06-23    |
+----+------------+--------------+--------------+
1 row in set (0.01 sec)

-- 4. Ambil 2 orang berikutnya lagi; kali ini hasilnya kosong
mysql> SELECT id, nama_depan, nama_belakang FROM orang
       ORDER BY id LIMIT 2 OFFSET 6;
Empty set (0.01 sec)

-- 5. Ambil 3 orang termuda
SELECT * FROM orang ORDER BY tanggal_lahir LIMIT 3;

-- 6. Ambil 3 orang tertua. Ini sama dengan mengatakan: “ambil 3
--      orang _terakhir_ berdasarkan urutan kemudaan”.
SELECT * FROM orang ORDER BY tanggal_lahir DESC LIMIT 3;
```

Dari contoh terakhir, kita bisa melihat bahwa untuk mengambil *N* buah baris terakhir (di mana kita tidak bisa menyebutkan OFFSET karena jumlah baris tidak diketahui) maka kita bisa membalikkan urutan penyortiran dan mengambil *N* baris *pertama*.

6.7 Fungsi-Fungsi Untuk Digunakan Di SELECT: Fungsi String

Salah satu yang membuat MySQL amat nikmat digunakan adalah tersedianya banyak fungsi builtin untuk macam-macam keperluan, meskipun banyak yang tidak diatur dalam standar SQL. Di lain pihak, database open source lain seperti Interbase/Firebird cukup minim fungsi builtinnya dan Anda harus menambahkan sendiri librari UDF (user-defined function) untuk mengadakan fungsi-fungsi seperti manipulasi string, enkripsi password, dan lain sebagainya.

Empat subbab mulai dari subbab ini akan secara sekilas membahas fungsi-fungsi apa saja yang cukup sering digunakan. Untuk daftar fungsi yang lengkap, silakan merujuk pada manual MySQL.

Rata-rata fungsi jika diberi masukan NULL akan menghasilkan NULL juga, kecuali fungsi-fungsi yang khusus untuk memanipulasi NULL.

Subbab ini membahas fungsi-fungsi yang berkaitan dengan string.

Panjang String

Untuk mengetahui panjang string digunakan fungsi CHAR_LENGTH(*str*) atau OCTET_LENGTH(*str*). Fungsi LENGTH() adalah sinonim untuk OCTET_LENGTH(*str*). Apa perbedaan keduanya? CHAR_LENGTH() menghitung jumlah karakter dan ini umumnya yang kita maksud/kehendaki. OCTET_LENGTH() menghitung jumlah byte yang dibutuhkan untuk menampung string. Untuk bahasa Indonesia atau Inggris yang sepenuhnya menggunakan ASCII, jumlah byte selalu sama dengan jumlah karakter karena semua karakter menempati 1 byte saja. Untuk teks berbahasa Jepang atau Cina atau multibyte lain, keduanya dapat berbeda.

Fungsi-fungsi ini diatur dalam standar SQL.

Kode ASCII

Fungsi ASCII(*str*) mengembalikan kode ASCII sebuah karakter. Jika str lebih dari satu karakter panjangnya, yang dipakai adalah karakter pertama. Fungsi ORD(*str*) sama seperti ASCII(), tapi aware terhadap keberadaan karakter multibyte dan mengembalikan nilai 1-byte hingga 4-byte karakter yang bersangkutan. Karena bahasa Indonesia maupun Inggris hanya menggunakan kode ASCII, maka dalam hal ini ORD() dan ASCII() setara hasilnya.

Untuk menghasilkan karakter dengan kode ASCII tertentu, gunakan fungsi CHAR(*angka*).

Mengambil Substring

Fungsi MySQL LEFT(*str*, *jum*), MID(*str*, *pos*, *jum*), dan RIGHT(*str*, *jum*) dapat Anda gunakan untuk mengambil substring dari string. Fungsi ini mirip dengan yang dijumpai di Basic/VB. LEFT() untuk mengambil *jum* karakter dari awal, RIGHT() mengambil *jum* karakter dari akhir, MID() mengambil *jum* karakter mulai dari posisi *pos* (awal string = 1).
Cara yang sesuai dengan standar SQL adalah dengan menggunakan fungsi SUBSTRING(), yang juga didukung oleh MySQL.

```
-- Resep 6-7-1: Sintaks fungsi SUBSTRING()
SUBSTRING(str FROM pos)
SUBSTRING(str FROM pos FOR jum)
```

Definisi *pos* dan *jum* sama seperti pada fungsi MID().

Mencari Substring

Fungsi standar SQL POSITION(*substr* IN *str*) mencari apakah *substr* berada dalam *str*. Jika ada, hasilnya posisi pertama ditemukan (awal string = 1). Jika tidak ditemukan, hasilnya 0.

"Mencukur" String (Trimming)

Fungsi standar SQL TRIM() dapat Anda gunakan untuk menghilangkan karakter tertentu dari awal dan/atau akhir sebuah string.

```
-- Resep 6-7-2: Sintaks fungsi TRIM()
TRIM([BOTH | LEADING | TRAILING] [karakter] FROM str)
```

Jika *karakter* tidak disebutkan, defaultnya adalah spasi (' '). LEADING berarti menghilangkan karakter yang ada di awal, TRAILING berarti mencari di akhir, BOTH berarti awal dan akhir. Default-nya adalah BOTH.
Contoh:

```
mysql> SELECT TRIM(BOTH 'x' FROM 'xxxSQLx');
       -> SQL
```

Mensubstitusi String

Fungsi standar SQL OVERLAY() berguna untuk mengganti substring dari sebuah string dengan substring lain.

```
-- Resep 6-7-3: Sintaks fungsi OVERLAY()
OVERLAY(str PLACING pengganti FROM pos [FOR jum])
```

Sayangnya, OVERLAY() belum didukung MySQL. Sebagai gantinya, gunakan fungsi MySQL REPLACE(*str*, *substr*, *pengganti*). Perbedaannya, REPLACE() akan mencari semua *substr* dalam *str* dan menggantinya dengan *pengganti*.

Contoh:

```
mysql> SELECT REPLACE('Modem', 'M', 'Voice M');
       -> Voice Modem
```

Menggabungkan String

Untuk menggabungkan string, digunakan operator ||. Defaultnya, MySQL menggunakan operator ini untuk tujuan lain yaitu operator logika OR. Sebagai gantinya, gunakan fungsi CONCAT(*str1*, *str2*, ...). Terdapat pula fungsi CONCAT_WS(*sep*, *str1*, *str2*, ...). Perbedaannya, CONCAT_WS() mengizinkan kita menyebutkan string *sep* untuk menjadi penyambung antarstring. Contoh:

```
mysql> SELECT CONCAT('a', 'b', 'c');
       -> abc
mysql> SELECT CONCAT_WS('-', 'a', 'b', 'c');
       -> a-b-c
```

Huruf Besar dan Huruf Kecil

Gunakan LOWER(*str*) untuk mengkonversi string menjadi huruf kecil, UPPER(*str*) untuk menjadi huruf besar.

Pola LIKE dan Regex

Standar SQL telah sejak lama mendukung operator LIKE. Operator ini berguna seperti regex untuk mencocokkan string dengan pola. Hanya saja polanya amat sederhana. Karakter meta yang dikenali hanyalah '_' (garis bawah) untuk mencocokkan satu karakter apa saja, '%' (persen) untuk mencocokkan nol atau lebih karakter apa saja, dan '\' (backslash) untuk meng-escape '_' dan '%' agar menjadi literal biasa. Contoh:

```
mysql> SELECT 'satu' LIKE 'sat_';
       -> 1
mysql> SELECT 'satu' LIKE 'sa%';
       -> 1
mysql> SELECT 'satu' LIKE 'satu%';
       -> 1
mysql> SELECT 'satu' LIKE 'sa_';
       -> 0
mysql> SELECT 'satu' LIKE 'satux%';
       -> 0
mysql> SELECT 'satu' LIKE 'satu\%'; - escaping berarti literal %
       -> 0
mysql> SELECT 'satu%' LIKE 'satu\%';
       -> 1
```

Karena regex semakin popular, maka banyak database termasuk MySQL yang menambahkan fasilitas pencocokan regex. Standar SQL:1999 pun akhirnya

memperkenalkan operator SIMILAR yang menerima pola regex. Namun MySQL saat ini belum mendukung SIMILAR, melainkan mendukung REGEXP (atau sinonimnya RLIKE) untuk melakukan pencocokan regex. Contoh:

```
mysql> SELECT 'satu' RLIKE 's[aeiou]t[aeiou]';
       -> 1
mysql> SELECT 'xxxxsatuxxx' RLIKE 's[aeiou]t[aeiou]';
       -> 1
mysql> SELECT 'xxxsatuxxx' RLIKE '^s[aeiou]t[aeiou]$';
       -> 0
```

Pencocokan dilakukan secara case-insensitive. Untuk melakukan pencocokan yang membedakan huruf besar dan kecil, gunakan REGEXP BINARY:

```
mysql> SELECT 'Satu' RLIKE 's...';
       -> 1
mysql> SELECT 'Satu' RLIKE BINARY 's...';
       -> 0
```

6.8 Fungsi-Fungsi Untuk Digunakan Di SELECT: Fungsi Numerik

Konversi Basis Bilangan

Untuk mengkonversi sebuah bilangan bulat dari desimal ke oktal, gunakan OCT(*desimal*). Hasilnya adalah string. Untuk konversi ke heksadesimal, gunakan HEX(*desimal*). Untuk konversi ke biner, gunakan BIN(*desimal*). Untuk mengkonversi dari basis lain ke basis desimal, atau secara umum dari satu basis ke basis lain, bisa digunakan fungsi CONV(*str*, *basis_awal*, *basis_tujuan*). OCT(N) ekivalen dengan CONV(N, 10, 8), HEX(N) ekivalen dengan CONV(N, 10, 16), BIN(N) ekivalen dengan CONV(N, 10, 2).

Matematika

MySQL menyediakan berbagai fungsi matematika lain: trigonometri (SIN(), COS(), ATAN(), dan lain sebagainya), logaritma (LOG(), LOG10(), EXP(), POW()), PI() untuk nilai pi, konversi derajat dari/ke radian (RADIANS(), DEGREES()).

Random

Fungsi RAND() berguna untuk menghasilkan bilangan random antara 0 dan 1. Fungsi ini dapat menerima argumen opsional berupa angka yang akan dijadikan

seed (biji). Jika seed sama, maka urutan bilangan acak yang dihasilkan akan sama setiap kali. Jika seed tidak disebutkan maka akan dipilih seed acak.

ABS(), SIGN()

ABS() berguna untuk menghasilkan nilai absolut sebuah bilangan. ABS(*bil*) ekivalen dengan:

```
CASE bil > 0 WHEN 1 THEN bil ELSE -bil END
```

SIGN() mengembalikan tanda bilangan, 1 untuk positif, 0 jika 0, -1 untuk negatif. SIGN(*bil*) setara dengan:

```
CASE bil > 0
  WHEN 1 THEN 1
  ELSE CASE bil < 0
    WHEN 1 THEN -1
    ELSE 0
  END
END
```

Pembulatan

FLOOR(*bil*) membulatkan ke bawah. CEIL(*bil*) membulatkan ke atas. ROUND(*bil*[, *ketelitian*]) membulatkan menurut aturan pembulatan normal. ROUND() menerima argumen kedua yaitu *ketelitian* berupa jumlah angka desimal yang diinginkan. Defaultnya adalah 0. TRUNCATE(*bil*, *ketelitian*) sama seperti ROUND() tapi membulatkan dengan memotong, bukan menurut aturan pembulatan (dan argumen kedua bersifat wajib). Perhatikan bahwa *ketelitian* dapat bernilai negatif: -1 berarti bulatkan hingga puluhan terdekat, -2 ratusan terdekat, dan seterusnya.

Demonstrasi berbagai fungsi pembulatan:

```
CREATE TABLE angka2 (x DOUBLE NOT NULL);
INSERT INTO angka2 VALUES (1.49);
INSERT INTO angka2 VALUES (1.5);
INSERT INTO angka2 VALUES (1.7);
INSERT INTO angka2 VALUES (2.5);
INSERT INTO angka2 VALUES (-2.5);
INSERT INTO angka2 VALUES (-203.4777);
INSERT INTO angka2 VALUES (153.6555);
```

```
mysql> SELECT
                        x,
         FLOOR(x)       AS f,
         CEIL(x)        AS c,
         ROUND(x)       AS r0,
         ROUND(x,1)     AS r1,
         ROUND(x,-1)    AS rmin1,
         TRUNCATE(x,0)  AS t0,
         TRUNCATE(x,1)  AS t1
       FROM angka2;
+----------+------+------+------+-------+-------+-----+--------+
| x        | f    | c    | r0   | r1    | rmin1 | t0  | t1     |
+----------+------+------+------+-------+-------+-----+--------+
|     1.49 |    1 |    2 |    1 |   1.5 |     0 |   1 |    1.4 |
|      1.5 |    1 |    2 |    2 |   1.5 |     0 |   1 |    1.5 |
|      1.7 |    1 |    2 |    2 |   1.7 |     0 |   1 |    1.6 |
|      2.5 |    2 |    3 |    2 |   2.5 |     0 |   2 |    2.5 |
|     -2.5 |   -3 |   -2 |   -2 |  -2.5 |    -0 |  -2 |   -2.5 |
| -203.4777 | -204 | -203 | -203 | -203.5 |  -200 | -203 | -203.4 |
|  153.6555 |  153 |  154 |  154 |  153.7 |   150 |  153 |  153.6 |
+----------+------+------+------+-------+-------+-----+--------+
7 rows in set (0.00 sec)
```

6.9 Fungsi-Fungsi Untuk Digunakan Di SELECT: Fungsi Tanggal dan Waktu

Perlu kita ingat kembali Bab 3, tipe data tanggal/waktu di MySQL saat ini cukup menyedihkan dan tidak sesuai standar SQL. Kelemahan utama adalah: 1) mengizinkan tanggal invalid masuk; 2) tidak ada dukungan untuk menyimpan ketelitian di bawah 1 detik; 3) tidak ada dukungan untuk menyimpan zona waktu; 4) tidak ada tipe data interval.

Tapi untuk dukungan fungsi-fungsi manipulasi tanggal/waktu itu sendiri, di MySQL tersedia cukup lengkap. Berikut sekilas pembahasannya:

Mengambil Elemen Tanggal/Waktu

MySQL menyediakan fungsi-fungsi YEAR(), MONTH(), DAY(), HOUR(), MINUTE(), SECOND() untuk mengekstrak elemen dari sebuah tanggal/waktu. Selain enam ini, bisa diekstrak juga WEEKDAY() yang menghasilkan 0 untuk Senin s.d. 6 untuk Sabtu. WEEKOFYEAR() yang mengembalikan nilai 1 s.d. 53,

MONTHNAME() untuk mengembalikan `'January'`, `'February'`, dan lain sebagainya. QUARTER() untuk mengembalikan 1 s.d. 4 (kuartal 1 hingga 4), dan lain sebagainya.

Fungsi standar SQL untuk mengekstrak elemen tanggal/waktu adalah EXTRACT():

```
-- Resep 6-9-1: Mengekstrak elemen tanggal/waktu
EXTRACT(elemen FROM tanggal)
```

elemen adalah literal berupa YEAR, MONTH, HOUR, dan lain sebagainya.

Aritmetika Tanggal/Waktu

Untuk menjumlah atau mengurangkan tanggal/waktu, kita dapat terlebih dahulu mengubah sebuah tanggal menjadi angka serial atau menjadi satuan terkecilnya. Untuk melakukannya, dapat digunakan fungsi-fungsi seperti TO_DAYS(), UNIX_TIMESTAMP(), dan TIME_TO_SEC(). TO_DAYS(*tgl*) mengkonversi tanggal menjadi jumlah hari sejak tahun 0. UNIX_TIMESTAMP(*tglwkt*) mengubah tanggal menjadi angka 32bit timestamp Unix, yang didefinisikan sebagai jumlah detik sejak 1 Jan 1970 00:00 UTC (dengan kata lain, sejak 1 Jan 1970 07:00 WIB). TIME_TO_SEC(*wkt*) mengubah waktu menjadi jumlah detik sejak pukul 00:00:00. Barulah setelah diubah ke dalam sebuah bilangan serial, kita dapat menambah/menguranginya dengan bilangan lain, misalnya menambah dengan X hari atau Y detik. Setelah aritmetika selesai, kita dapat konversi lagi ke tipe tanggal/waktu menggunakan fungsi FROM_DAYS(), FROM_UNIXTIME(), dan SEC_TO_TIME().

Cara di atas biasa memang dipakai di dalam bahasa pemrograman, tapi kadang terasa terlalu merepotkan. MySQL menyediakan juga fungsi DATE_ADD() dan DATE_SUB() untuk melakukan perhitungan langsung. Contoh:

```
-- menambah satu hari
DATE_ADD(tgl, INTERVAL 1 DAY);

-- 3 jam lalu
DATE_SUB(NOW(), INTERVAL 3 HOUR);
```

CURRENT_TIME(), CURRENT_TIMESTAMP()/NOW()

CURRENT_TIME() mengembalikan waktu saat ini (mis: '12:00:45'), tipe datanya TIME. CURRENT_DATE() mengembalikan tanggal saat ini (mis: '2004-09-15', tipe data DATE. CURRENT_TIMESTAMP() mengembalikan timestamp saat ini (mis: '2004-09-15 12:00:45'), tipe datanya DATETIME. NOW() adalah sinonim untuk CURRENT_TIMESTAMP().

Pemformatan Tanggal

DATE_FORMAT(*tgl*, *strformat*) dapat digunakan untuk menghasilkan string yang berisi tanggal yang telah diformat sesuai spesifikasi yang kita inginkan. Fungsi ini mirip seperti `date()` di PHP atau `strftime()` di C. Format lengkap dapat dilihat di manual MySQL. Contoh (dicomot dari manual MySQL):

```
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%W %M %Y');
       -> 'Saturday October 1997'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%H:%i:%s');
       -> '22:23:00'
```

Zona Waktu

Untuk mengetahui timezone saat ini di MySQL, dapat digunakan perintah:

```
— Resep 6-9-2: Mengetahui zona waktu saat ini di MySQL
SHOW VARIABLES LIKE 'system_time_zone';
```

Untuk mengkonversi tanggal/tanggal-waktu dari satu zona waktu ke zona waktu lain, gunakan fungsi CONVERT_TZ(*tgl*, *zona_asal*, *zona_tujuan*). Zona asal dan tujuan adalah simbol zona waktu yang terdaftar di OS Anda. Sebagai contoh, WIB (UTC+7) adalah "WIT" (West Indonesian Time).

6.10 Fungsi-Fungsi Untuk Digunakan Di SELECT: Fungsi Lain-Lain

Nilai Default: COALESCE()

COALESCE(*a*, *b*, *c*, ...) akan memilih nilai pertama yang bukan NULL. Jika *a* bukan NULL maka hasilnya *a*, jika *a* NULL maka *b* dites apakah NULL atau tidak. Jika NULL hasilnya *b*, dan seterusnya. Jika semua NULL, maka hasil akhir adalah NULL. COALESCE(*a*, *b*, *c*) ekivalen dengan:

```
CASE a IS NOT NULL
  WHEN 1 THEN a
  ELSE CASE b IS NOT NULL
    WHEN 1 THEN b
    ELSE c
  END
END
```

COALESCE() berguna untuk memberi nilai default jika sebuah kolom tidak diketahui nilainya (alias NULL). Contoh:

```
-- jika nomor hp tidak diketahui, gunakan nomor telepon rumah
--   jika keduanya tidak diketahui, gunakan '000-0000'
SELECT CONCAT(nomor_hp, telp, '000-0000') AS telp FROM kontak;
```

NULL Jika Sama: NULLIF()

NULLIF(*a*, *b*) akan menghasilkan NULL jika *a* = *b*. Jika tidak, *a* yang dikembalikan. Setara dengan:

```
CASE WHEN a = b THEN NULL ELSE a END
```

Sejauh ini, saya pribadi jarang sekali menggunakan NULLIF().

Konversi Tipe: CAST()

CASTT() juga merupakan salah satu fungsi standar SQL yang jarang digunakan di MySQL, dikarenakan sifat MySQL yang weakly-typed dan melakukan otokonversi tipe.CAST() berguna untuk mengubah sebuah tipe data menjadi tipe data lain.

```
-- Resep 6-10-1: Sintaks CAST()
CAST(ekspresi AS tipedata)
```

di mana *tipedata* adalah CHAR, INTEGER, dan lain sebagainya.

Password

Fungsi PASSWORD() dapat digunakan untuk membuat hash password sama seperti yang digunakan oleh MySQL sendiri untuk tabel usernya. Jika Anda membuat tabel *anggota* atau *user* dan perlu menyimpan password dalam bentuk hash, Anda tidak wajib menggunakan fungsi ini; bahkan saya sarankan menggunakan saja metode yang tidak spesifik MySQL seperti MD5() atau SHA1() (dengan ditambah salt).

Hashing

MySQL menyediakan metode hashing yang popular yaitu MD5() dan SHA1().

```
LAST_INSERT_ID()
```

Fungsi ini untuk mengembalikan nilai yang terakhir dibuat untuk kolom auto_increment. Contoh:

```
CREATE TABLE nama (
  i INT PRIMARY KEY AUTO_INCREMENT,
  nama VARCHAR(100)
);
INSERT INTO nama (nama) VALUES ('Andi');
SELECT LAST_INSERT_ID(); – hasilnya 1
INSERT INTO nama (nama) VALUES ('Budi');
SELECT LAST_INSERT_ID(); – hasilnya 2
```

6.11 Latihan SELECT

Setelah mengenal semua klausa dasar SELECT dan berbagai fungsi untuk mengolah berbagai tipe data, cobalah latihan-latihan berikut. Untuk latihan ini, data diambil dari file *karyawan.csv* yang dapat Anda ambil di CD buku direktori code/karyawan.csv. Jika Anda tidak memiliki CD buku, bisa mengambil file ini di http://groups.yahoo.com/group/dianrakyat-buku-pemrograman/files/bukuresepsql/. Data ini berisi daftar nama, jabatan, dan gaji karyawan sebuah perusahaan dan akan digunakan untuk mengisi tabel *karyawan* yang strukturnya sebagai berikut:

```
CREATE TABLE karyawan (
  id INT PRIMARY KEY,
  nama_depan VARCHAR(64) NOT NULL,
  nama_belakang VARCHAR(64) NOT NULL,
  kelamin CHAR(1) NOT NULL,
  tempat_lahir VARCHAR(64) NOT NULL,
  tgl_lahir DATE NOT NULL,
  tgl_masuk DATE NOT NULL,
  divisi VARCHAR(32) NOT NULL,
  level VARCHAR(32) NOT NULL,
  jabatan VARCHAR(64) NOT NULL,
  gaji NUMERIC(18,4) NOT NULL
);
```

Penjelasan struktur tabel: *tgl_masuk* adalah tanggal si karyawan terdaftar di perusahaan tersebut. *Divisi* adalah pembagian utama karyawan, seperti sales, marketing, HRD, keuangan, dan lain sebagainya. Ada karyawan tertentu yang tidak masuk ke dalam salah satu divisi, yaitu direktur utama (beserta wakil dan sekretarisnya) atau karyawan urusan umum seperti satpam dan office boy. *level* mencatat tingkat jabatan karyawan, mulai dari nonstaf (pekerja lepas atau kontrak), klerek (staf tingkat bawah, umumnya bukan lulusan sarjana), staf, manajemen (manajer ke atas). *gaji* mencatat gaji saat ini.

Soal

1. Masukkanlah data dari file .CSV tersebut di atas ke dalam baris-baris tabel *karyawan*.
2. Buat query untuk menjawab pertanyaan: "Ada berapa banyak divisi di perusahaan dan sebutkan?"
3. "Berapa jumlah karyawan per divisi? Divisi apakah yang terbanyak karyawannya? Tersedikit?"
4. Daftarkan nama, divisi, dan jabatan tiap karyawan pria yang telah bekerja di perusahaan selama lebih dari 4 tahun.
5. Perusahaan mengadakan undian berhadiah. Pilihlah 5 karyawan secara acak untuk menentukan siapa pemenangnya.
6. Berapa jumlah staf per kelompok usia? Kelompok usia yang diinginkan: 25 tahun ke bawah, 26-30, 31-35, 36-40, 41-45, 46-50, 51-55, 56-60, dan di atas 60 tahun.
7. Untuk tiap divisi, tampilkan berapa angka perbandingan jumlah karyawan tetap staf nonmanajemen (staf biasa, klerek) : jumlah manajemen (manajer ke atas). Angka 2 berarti staf 2x lebih banyak dari manajemen. Catatan: untuk divisi yang tidak memiliki staf atau manajemen, jangan ikutsertakan.
8. Divisi mana yang rata-rata gaji stafnya paling besar? Paling kecil?
9. Divisi mana yang paling banyak lelakinya *dibandingkan* perempuan? (Tentu saja, divisi yang semuanya lelaki berarti *paling* banyak lelakinya).
10. Berapa gaji total yang harus dikeluarkan perusahaan untuk menggaji semua karyawan (per divisi dan level, serta total)?
11. Berapa gaji karyawan pria yang paling tua di divisi marketing?

Jawaban Latihan

Untuk pertanyaan #1, urutan kolom-kolom pada file karyawan.csv kebetulan sudah sama dengan urutan kolom tabel:

```
id,nama_depan,nama_belakang,kelamin,tempat_lahir,tgl_lahir,
tgl_masuk,divisi,level,jabatan,gaji
1,Koko,Wacana,L,Bekasi,10/4/1953,12/19/1994,,direksi,Direktur
Utama,24000000
...
```

tapi masalahnya, format tanggalnya tidak bisa diterima langsung oleh MySQL. MySQL mengharapkan literal tanggal dalam format `'YYYY-MM-DD'`, bukan `'MM/DD/YYYY'`.

Untuk menyelesaikan ini, sebetulnya dengan bantuan utilitas teks seperti sed, perl, bahkan php pun bisa. Tapi mari kita mencoba menggunakan MySQL, sekaligus mendemonstrasikan pengolahan teks dan tanggal menggunakan MySQL.

Pertama, kita tampung hasil CSV ke tabel sementara yang katakanlah bernama *karyawan_tmp*. Struktur tabel sementara ini sama dengan tabel *karyawan*, dengan perbedaan `tgl_lahir` dan `tgl_masuk` didefinisikan sebagai VARCHAR(10).

```
CREATE TABLE karyawan_tmp (
  id INT PRIMARY KEY,
  nama_depan VARCHAR(64) NOT NULL,
  nama_belakang VARCHAR(64) NOT NULL,
  kelamin CHAR(1) NOT NULL,
  tempat_lahir VARCHAR(64) NOT NULL,
  tgl_lahir VARCHAR(10) NOT NULL,
  tgl_masuk VARCHAR(10) NOT NULL,
  divisi VARCHAR(32) NOT NULL,
  level VARCHAR(32) NOT NULL,
  jabatan VARCHAR(64) NOT NULL,
  gaji NUMERIC(18,4) NOT NULL
);
```

Impor data dari file .CSV menggunakan perintah LOAD DATA INFILE. Taruh dulu file .CSV ke mesin yang sama dengan mesin tempat `mysqld` berjalan, misalnya di `/tmp/karyawan.csv` (di Windows Anda bisa menggunakan mis: `C:/karyawan.csv`) Lalu:

```
LOAD DATA INFILE '/tmp/karyawan.csv'
  INTO TABLE karyawan_tmp
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY
'\\'
  IGNORE 1 LINES;
```

Hasilnya, jika sukses, adalah 70 baris ditambahkan ke tabel *karyawan_tmp*. Jika gagal, coba cek apakah file lokasi dan permissionnya sudah benar. Jika Anda tidak memiliki akses ke mesin server, bisa juga Anda gunakan LOAD LOCAL DATA INFILE dan menaruh file .CSV-nya di komputer Anda sendiri. Lihat Bab 5 atau manual MySQL untuk detilnya.

Setelah *karyawan_tmp* terisi, kini tinggal masalah mengkonversi tanggal dan memasukkan ke tabel tujuan akhir *karyawan*:

```
INSERT INTO karyawan
  SELECT
    id, nama_depan, nama_belakang, kelamin, tempat_lahir,
```

```
    CONCAT(
      RIGHT(tgl_lahir, 4), – tahun
      '-',
      SUBSTRING_INDEX(tgl_lahir, '/', 1), – bulan
      '-',
      SUBSTRING_INDEX(SUBSTRING_INDEX(tgl_lahir,'/',-2),'/',1) –
tgl
    ) AS tgl_lahir,
    CONCAT(
      RIGHT(tgl_masuk, 4), – tahun
      '-',
      SUBSTRING_INDEX(tgl_masuk, '/', 1), – bulan
      '-',
      SUBSTRING_INDEX(SUBSTRING_INDEX(tgl_masuk,'/',-2),'/',1) –
tgl
    ) AS tgl_masuk,
    divisi, level, jabatan, gaji
  FROM karyawan_tmp;
```

Mari perhatikan perintah di atas. Kita melakukan INSERT ... SELECT untuk memasukkan baris-baris hasil query ke dalam tabel *karyawan*. Query dari tabel *karyawan_tmp* sifatnya cukup sederhana; kita mendaftar semua kolom dalam urutan yang sama, kecuali mengganti dua kolom *tgl_lahir* dan *tgl_masuk* dengan ekspresi. Kita mengekstrak tahun, lalu bulan, lalu tanggal dan menggabungkannya dengan CONCAT() sehingga menjadi berformat `'YYYY-MM-DD'` dan siap diterima kolom bertipe DATE. Untuk mengekstrak substring kita menggunakan fungsi MySQL SUBSTRING_INDEX(*str*, *pembatas*, *jum*) untuk mengambil substring dari *str* sebelum ditemukannya *jum* buah *pembatas*. Contoh,
`SUBSTRING_INDEX('2004-09-30', '-', 1)` akan bernilai `2004`,
`SUBSTRING_INDEX('2004-09-30', '-', 2)` bernilai `'2004-09'`, dan seterusnya.

Terakhir, jika sudah tidak diperlukan kita bisa menghapus tabel sementara kita:

```
DROP TABLE karyawan_tmp;
```

Jawaban pertanyaan #2:

```
SELECT DISTINCT divisi FROM karyawan WHERE divisi <> '';
```

Hasilnya adalah 7 divisi. Divisi yang bernilai string kosong (`''`) tidak kita ikut sertakan.

Jawaban pertanyaan #3:

```
mysql> SELECT divisi, COUNT(*) jum_karyawan FROM karyawan
       WHERE divisi <> ''
       GROUP BY divisi ORDER BY jum_karyawan;
+-----------+-------------+
| divisi    | jum_karyawan |
+-----------+-------------+
| HRD       |            4 |
| R&D       |            4 |
| teknis    |            7 |
| keuangan  |            7 |
| IT        |            8 |
| sales     |           14 |
| marketing |           16 |
+-----------+-------------+
7 rows in set (0.00 sec)
```

Hasilnya, divisi HRD dan R&D sama-sama hanya memiliki 4 karyawan sementara divisi marketing karyawannya paling banyak, 16 orang. Perusahaan ini nampaknya berorientasi pada sales dan marketing.

Jawaban pertanyaan #4:

```
SELECT id, nama_depan, nama_belakang, divisi, jabatan FROM karyawan
WHERE
  kelamin = 'L' AND
  tgl_masuk < DATE_SUB(CURRENT_DATE(), INTERVAL 4 YEAR);
```

Hasilnya adalah 12 karyawan. Untuk mencari yang telah bekerja lebih dari 4 tahun, kita membentuk dulu tanggal pada 4 tahun lalu dengan mengurangi tanggal hari ini dengan 4 tahun. Lalu membandingkannya dengan *tgl_masuk* (jika lebih kecil, berarti tanggal yang lebih tua). Tip: Anda dapat menyertakan dulu kolom *kelamin* dan *tgl_masuk* dalam eksprresi SELECT untuk mengecek apakah query Anda benar.

Jawaban pertanyaan #5:

```
-- Resep 6-11-1: Sampling N buah baris acak dari tabel
SELECT ... FROM nama_tabel ORDER BY RAND() LIMIT N;
```

Masih ingat dengan ORDER BY RAND() di Resep 6-3-2 untuk menampilkan baris tabel secara acak? Kita hanya tinggal menambahkan klausa LIMIT untuk mengambil *N* buah baris saja. Dalam kasus kita kali ini:

```
-- Nonstaf tidak diikutsertakan, hanya karyawan tetap saja
SELECT * FROM karyawan WHERE level <> 'nonstaf'
ORDER BY RAND() LIMIT 5;
```

Jawaban pertanyaan #6: Masalah ini merupakan masalah custom grouping, yakni mengelompokkan berdasarkan nilai/ekspresi tertentu dan bukan berdasarkan kolom tertentu. Jadi kita membuat dulu ekspresinya (lihat *kelompok_usia* di query di bawah).

```
SELECT
  IF(tgl_lahir >= DATE_SUB(CURRENT_DATE(),
                           INTERVAL 25 YEAR),
    '25 tahun ke bawah',
    IF(tgl_lahir >= DATE_SUB(CURRENT_DATE(),
                             INTERVAL 30 YEAR),
      '26-30',
      IF(tgl_lahir >= DATE_SUB(CURRENT_DATE(),
                               INTERVAL 35 YEAR),
        '31-35',
        IF(tgl_lahir >= DATE_SUB(CURRENT_DATE(),
                                 INTERVAL 40 YEAR),
          '36-40',
          IF(tgl_lahir >= DATE_SUB(CURRENT_DATE(),
                                   INTERVAL 45 YEAR),
            '41-45',
            IF(tgl_lahir >= DATE_SUB(CURRENT_DATE(),
                                     INTERVAL 50 YEAR),
              '46-50',
              IF(tgl_lahir >= DATE_SUB(CURRENT_DATE(),
                                       INTERVAL 55 YEAR),
                '51-55',
                IF(tgl_lahir >= DATE_SUB(CURRENT_DATE(),
                                         INTERVAL 60 YEAR),
                '56-60',
                'di atas 60 tahun'
                )
              )
```

```
            )
          )
        )
      )
    )
  ) AS kelompok_usia,
  COUNT(*)
FROM karyawan
GROUP BY kelompok_usia;
```

Kita menggunakan serangkaian IF() untuk membentuk beberapa kelompok usia, berdasarkan tanggal lahir tentunya. Tip: Sewaktu mengetes, hilangkan dulu klausa GROUP BY dan tambahkan kolom *tgl_lahir* di ekspresi SELECT untuk melihat apakah setiap tanggal lahir telah dikategorikan ke dalam kelompok usia yang tepat. Dan untuk mengetes apakah semua karyawan telah masuk ke dalam sebuah kelompok usia, Anda bisa menambahkan opsi WITH ROLLUP pada GROUP BY. Contoh hasilnya (dengan WITH ROLLUP):

```
+------------------+----------+
| kelompok_usia    | COUNT(*) |
+------------------+----------+
| 25 tahun ke bawah |       13 |
| 26-30            |       26 |
| 31-35            |       14 |
| 36-40            |        8 |
| 41-45            |        4 |
| 46-50            |        3 |
| 51-55            |        2 |
| NULL             |       70 |
+------------------+----------+
8 rows in set (0.00 sec)
```

Jawaban pertanyaan #7: Pertanyaan ini mendemonstrasikan kegunaan klausa

```
HAVING:
SELECT
  divisi,
  SUM(IF(level='staf' OR level='klerek',1,0)) AS jum_staf,
  SUM(IF(level='manajemen' OR level='direksi',1,0)) AS
jum_manajemen,
  SUM(IF(level='staf' OR level='klerek',1,0)) /
```

```
    SUM(IF(level='manajemen' OR level='direksi',1,0)) AS rasio
FROM karyawan
WHERE divisi <> ''
GROUP BY divisi
HAVING jum_staf > 0 AND jum_manajemen > 0;
```

Jawaban pertanyaan #8:

```
SELECT
  divisi, ROUND(AVG(gaji),-3) as gaji_rata2
FROM karyawan
WHERE level='staf' AND divisi <> ''
GROUP BY divisi
ORDER BY gaji_rata2 DESC;
```

Di sini saya membulatkan gaji rata-rata ke ribuan terdekat menggunakan ROUND(*x*, -3). Untuk mengetahui perbandingan gaji manajer antardivisi, cukup ganti kondisi `level='staf'` menjadi `level='manajemen'`.

Jawaban pertanyaan #9:

```
SELECT
  divisi,
  SUM(IF(kelamin='L',1,0)) as jum_lelaki,
  SUM(IF(kelamin='P',1,0)) as jum_perempuan,
  IF(SUM(IF(kelamin='P',1,0)) = 0,
     SUM(IF(kelamin='L',1,0))*1000000,
     SUM(IF(kelamin='L',1,0))/
       SUM(IF(kelamin='P',1,0))
  ) AS rasio
FROM karyawan
WHERE divisi <> ''
GROUP BY divisi
ORDER BY rasio DESC;
```

Pertanyaan ini mirip dengan #7 yaitu mengurutkan berdasarkan rasio. Hanya saja, jika jumlah karyawan wanita tidak ada (pembagi = 0) maka kita menggunakan angka yang besar (1 juta * jumlah karyawan pria), agar kolom *rasio* tetap bisa diurutkan. Hasilnya, divisi IT menduduki urutan pertama lahan yang "gersang wanita" dengan 8 karyawan pria tanpa karyawan wanita, disusul divisi teknis dengan 7 : 0. Yang paling seimbang atau lebih banyak wanitanya adalah divisi keuangan.

Jawaban pertanyaan #10:

```
SELECT divisi, level, SUM(gaji) FROM karyawan
GROUP BY divisi,level WITH ROLLUP;
```

Hasilnya 207,55 juta rupiah total per bulan.

Jawaban pertanyaan #11:

```
SELECT gaji FROM karyawan WHERE divisi='marketing'
ORDER BY tgl_lahir ASC LIMIT 1;
```

Hasilnya: Rp 4,5 juta (gaji Sdr. Sjarif Tegar, yang lahir pada tahun 1956.)